Watch and Learn: z/VM CMS Pipelines

Move closer to front

No

Can you read this?

Yes

Ok

Bill Bitner
bitnerb@us.ibm.com
Notice Regarding Specialty Engines (e.g., zIIPs, zAAPs and IFLs):

Any information contained in this document regarding Specialty Engines ("SEs") and SE eligible workloads provides only general descriptions of the types and portions of workloads that are eligible for execution on Specialty Engines (e.g., zIIPs, zAAPs, and IFLs). IBM authorizes customers to use IBM SE only to execute the processing of Eligible Workloads of specific Programs expressly authorized by IBM as specified in the “Authorized Use Table for IBM Machines” provided at [www.ibm.com/systems/support/machine_warranties/machine_code/aut.html](http://www.ibm.com/systems/support/machine_warranties/machine_code/aut.html) ("AUT").

No other workload processing is authorized for execution on an SE.

IBM offers SEs at a lower price than General Processors/Central Processors because customers are authorized to use SEs only to process certain types and/or amounts of workloads as specified by IBM in the AUT.
The Basics

- CMS Pipelines is a programming framework that is very powerful. Like its name, and unix pipes, it allows data to flow through different pipes, connectors, filters, etc. We typically call each section of Pipelines a “Stage”.
  - Program
  - Arguments
  - Stage Separator

- Sources of data:
  - Constants
  - Files
  - Xedit
  - Other pipes
  - z/VM system services

- Successful programmers use “Pipe Think”
  - Breaking down the problem into various steps in manipulating the data

- Most common separator or connector is the vertical bar “|”
CMS Pipelines Implementation and Terminology

- CMS command called PIPE
  - Pipeline scanner analyses the pipeline specification
    - Multiple “stages” separated by a “stage separator” (the pipe character)
    - Each stage specifies the program and its arguments
  - Pipeline dispatcher invokes programs as specified by the pipeline
    - Not ordinary CMS programs, but specifically designed for use in pipes
    - CMS Pipelines built-in programs and user-written programs
    - Runs the programs while pumping the data through the pipeline
Everyone’s First Program

- Simple example:
  - Pipe Literal Hello World | Cons
    - "Literal": means whatever follows is data for pipe
    - "Cons": direct data to the console

- Some output doesn’t delay, e.g.
  - Pipe Literal Hello World | cons | > Hello World A

```
pipe literal Hello World | cons
Hello World
Ready;
```
Help Me!

- Help Pipe
  - Basic control on the Pipelines command

- Help Pipe Menu
  - Gives menu listing the various stage programs

- Help pipe stage
  - Help on individual stage
  - Includes examples!

- Reference Book includes help plus lists related stages
CP Command Output in Pipelines

- Use the “CP” stage followed by a command

```bash
pipe cp query time | cons
TIME IS 10:48:19 EDT MONDAY 10/09/17
CONNECT= 99:59:59 VIRTCPU= 000:01.61 TOTCPU= 000:02.47
Ready;
```

- CP Syntax
  - Number deals with how big a buffer to create for commands with very large output
Problem 1: I Want to know how many Page Volumes I Have?

- I could use QUERY ALLOC PAGE and count, but I don’t like to count

<table>
<thead>
<tr>
<th>VOLID</th>
<th>RDEV</th>
<th>EXTENT</th>
<th>EXTENT</th>
<th>TOTAL</th>
<th>PAGES</th>
<th>HIGH</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>START</td>
<td>END</td>
<td>PAGES</td>
<td>IN USE</td>
<td>PAGE USED</td>
<td></td>
</tr>
<tr>
<td>-------</td>
<td>------</td>
<td>--------</td>
<td>--------</td>
<td>-------</td>
<td>---------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>PGG700</td>
<td>2700</td>
<td>1</td>
<td>10016</td>
<td>1761K</td>
<td>76764</td>
<td>165699  4%</td>
<td></td>
</tr>
<tr>
<td>PGG701</td>
<td>2701</td>
<td>1</td>
<td>10016</td>
<td>1761K</td>
<td>82926</td>
<td>179154  4%</td>
<td></td>
</tr>
<tr>
<td>PGG702</td>
<td>2702</td>
<td>1</td>
<td>10016</td>
<td>1761K</td>
<td>79160</td>
<td>174218  4%</td>
<td></td>
</tr>
<tr>
<td>PGG703</td>
<td>2703</td>
<td>1</td>
<td>10016</td>
<td>1761K</td>
<td>83376</td>
<td>181899  4%</td>
<td></td>
</tr>
<tr>
<td>PGG704</td>
<td>2704</td>
<td>1</td>
<td>10016</td>
<td>1761K</td>
<td>82471</td>
<td>174240  4%</td>
<td></td>
</tr>
<tr>
<td>PGG705</td>
<td>2705</td>
<td>1</td>
<td>10016</td>
<td>1761K</td>
<td>80245</td>
<td>168644  4%</td>
<td></td>
</tr>
<tr>
<td>PGG706</td>
<td>2706</td>
<td>1</td>
<td>10016</td>
<td>1761K</td>
<td>87295</td>
<td>186236  4%</td>
<td></td>
</tr>
<tr>
<td>PGG707</td>
<td>2707</td>
<td>1</td>
<td>10016</td>
<td>1761K</td>
<td>77462</td>
<td>177829  4%</td>
<td></td>
</tr>
<tr>
<td>PGG708</td>
<td>2708</td>
<td>1</td>
<td>10016</td>
<td>1761K</td>
<td>83676</td>
<td>187200  4%</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PGG734</td>
<td>2722</td>
<td>1</td>
<td>10016</td>
<td>1761K</td>
<td>84817</td>
<td>176826  4%</td>
<td></td>
</tr>
</tbody>
</table>

**SUMMARY**
61622K  2817K  4%
**USABLE**
61622K  2817K  4%

Ready;
Problem 1: I Want to know how many Page Volumes I Have?

- I could then look in the file ALLOC PAGE A and count, but I don’t like to count that either.

```
PIPE CP q alloc page | > alloc page a
Ready;
```

- Wait, that’s wrong, I forgot about the headers and summary lines, we have to remove 3 from top and 3 from bottom.

```
PIPE CP q alloc page | COUNT lines | CONS
41
Ready;
```

```
PIPE CP q alloc page | DROP 3 | DROP LAST 3 | COUNT LINES | CONS
35
Ready;
```
Problem 2: Which real volumes contain my virtual disks

- I could use QUERY DASD and cut and paste, but I am lazy

```
QUERY V DASD
DASD 009B 3390 USE724 R/O    10 CYL ON DASD D666 SUBCHANNEL = 0017
DASD 0120 3390 SYE711 R/O    250 CYL ON DASD D548 SUBCHANNEL = 0016
DASD 0190 3390 USG7CB R/O    214 CYL ON DASD 2524 SUBCHANNEL = 000F
DASD 0191 3390 USG72A R/W    300 CYL ON DASD D57D SUBCHANNEL = 0008
DASD 019B 3390 USE740 R/O    300 CYL ON DASD E360 SUBCHANNEL = 0012
DASD 019D 3390 US7E53 R/O    250 CYL ON DASD E375 SUBCHANNEL = 0010
DASD 019E 3390 USG7AO R/O    400 CYL ON DASD DB3F SUBCHANNEL = 0011
DASD 019F 3390 USE73L R/O    100 CYL ON DASD D76E SUBCHANNEL = 0013
DASD 01A1 3390 US7EA6 R/O    100 CYL ON DASD C703 SUBCHANNEL = 0015
DASD 0223 3390 USP749 R/W    22 CYL ON DASD D60F SUBCHANNEL = 0000
  ...  DASD 02BD 3390 USP773 R/W  2000 CYL ON DASD D50B SUBCHANNEL = 0005
DASD 0399 3390 USP749 R/O    30 CYL ON DASD D60F SUBCHANNEL = 0014
DASD 0419 3390 USE71F R/W    17 CYL ON DASD D748 SUBCHANNEL = 0003
DASD 0A91 3390 USE719 R/O    10 CYL ON DASD D745 SUBCHANNEL = 0018
Ready;
```
Problem 2: Which real volumes contain my virtual disks

- Let’s use Pipelines and a new very powerful stage called “SPEC”
- SPEC has many options, one is to parse different words
  - Here we take the 10th word and place it in column 1
  - The real device address was the 10th word

```
PIPE CP QUERY V DASD | SPEC w10 1 | cons
D666
D548
2524
D57D
E360
E375
DB3F
D76E
C703
D60F
...
D50B
D60F
D748
D745
Ready;
```
Problem 2: Which real volumes contain my virtual disks

- How many have duplicate Volumes? Lets use SPEC and COUNT to determine total number of virtual DASD
  
  ```
  PIPE CP QUERY V DASD | SPEC w10 | COUNT LINES | CONS
  21
  Ready;
  ```

- Now use a new SORT option to only get the UNIQUE ones
  
  ```
  PIPE CP QUERY V DASD | SPEC w10 | SORT UNIQUE | COUNT LINES | CONS
  20
  ```

- So there is one real device that has two virtual DASD on it (21 – 20 = 1)
Problem 2: Which real volumes contain my virtual disks

- What if we want more details? Use SORT COUNT.

<table>
<thead>
<tr>
<th>PIPE</th>
<th>CP</th>
<th>QUERY DA</th>
<th>SPEC w10 2</th>
<th>SORT COUNT</th>
<th>CONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>C703</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>DA15</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>DB3F</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>D44A</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>D50B</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>D548</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>D57D</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>D60F</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>D617</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>D664</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>D666</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>E30F</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>E360</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>E370</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2524</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- One Volume (D60F) has two vdevs on it, rest have one

- Notice:
  - Moved w10 from column 1 in previous example to column 2 in order to make more readable here
  - SORT COUNT – sorts the input records and in the process removes duplicates. The COUNT option of SORT keeps count of total records matching this.
Problem 2: Which real volumes contain my virtual disks

- But I don’t want this on my console, especially for large machines so I will put in a CMS file

```
PIPE CP QUERY V DASD | SPEC w10 2 | SORT COUNT | > virtreal dasdlist a
```

- The CONS stage is replaced the “>” which indicates output goes to file id that follows, in this case “virtreal dasdlist a” file

- You can use both CONS and a “>” director. I do this sometimes for validation

```
PIPE CP QUERY V DASD | SPEC w10 2 | SORT COUNT | > virtreal dasdlist a | CONS
```

- “>” creates a new file of that name even if one exists
- “>>” appends to file of that name if it exists, otherwise creates new one
- “<“ allows you to read from a file on other end of pipe
Use as a REXX Exec

/* BITQVIR EXEC */
/* Bit's Virtual on Real Dasd List Exec */
'PIPE CP QUERY V DASD', /* Get list of virtual DASD */
'| SPEC W10 2', /* Real address is word 10 */
'| SORT COUNT', /* Find duplicates */
'| SORT 1-10 DESCEND', /* Sort descendig on count */
'| > virtreal dasdlist a' /* Write out results */
exit

• Use continuation character, the comma.
• Start with a connector
• Use comments
• Introduced new SORT stage with DESCEND option to sort in descending order based on columns 1-10
Two Nice features for REXX – First VAR

/* BITQDVAR - Example put number of Devices in a variable */
'PIPE CP QUERY V DASD', /* Get list of virtual DASD */
'| SPEC W10 2', /* Real address is word 10 */
'| SORT COUNT', /* Find duplicates */
'| COUNT LINES', /* Count of real devices */
'| VAR num_real_devices' /* store in variable */
Say "Number Real:" num_real_devices /* now can use as variable */

exit

BITQDVAR
Number Real: 20
Ready;
Two Nice features for REXX – Second STEM

`/* BITQDSTM - REXX Stem variable example with Real device addresses */`

`'PIPE CP QUERY V DASD',               /* Get list of virtual DASD */`
`'| SPEC W10 1',                       /* Real address is word 10 */`
`'| STEM' real_device.                 /* Put the lines into a Stem */`

Do i = 1 to real_device.0            /* .0 is number of entries */
    Say 'A device is:' real_device.i /* Do something with it */
End /* For each real device */

exit
As in real life, plumbing often involves more than a single straight path.
Multi-streams – A few Things to Know

- Most often done inside an Exec.
- Need a way to mark the end of streams
  - PIPE (endchar ?)
- Need a way to mark where streams connect
  - Labels, for simple pipes a character followed by colon (e.g. “f.”)
- Examples in the Help are your friend!
Problem Three: Determine Entitlement of a Logical Partition

- Without using performance data.
- Multiple ways to solve this
- Leverage, the CP command QUERY PROC TOPOLOGY

```
Q PROC TOPOLOGY
TOPOLOGY
   NESTING LEVEL: 02  ID: 01
   NESTING LEVEL: 01  ID: 01
   PROCESSOR 00  MASTER  CP  VH  0000
   PROCESSOR 01  ALTERNATE  CP  VH  0001
   PROCESSOR 02  ALTERNATE  CP  VM  0002
   PROCESSOR 03  ALTERNATE  CP  VL  0003

Ready;
```

- 2 Vertical High, 1 Vertical Medium, 1 Vertical Low
  - High = 100, Low = 0, Medium = something else
Problem Three: Determine Entitlement of a Logical Partition

- **Logic**
  - Find out how many vertical highs, how many vertical mediums, and how many vertical lows
  - Do math on those counts $100 \times VH + 75 \times VM + 0 \times VL$

- I can use a similar approach but I need to do it for all three (well really just VH and VM)

- Use LOCATE stage to find a record that contains a string

- Use VAR to save the counts and do the math
Problem Three: Determine Entitlement of a Logical Partition

/* Determine rough entitlement */
'PIPE (end ?)',
'| CP Q PROC TOPOLOGY', /* Get topology Info */
'| f: fanout', /* fanout to all the streams */
'| locate /VH/', /* locate vertical high */
'| count lines', /* count lines with them */
'| VAR VH', /* store count in variable VH */
'|?f:', /* second stream */
'| locate /VM/', /* locate vertical medium */
'| count lines', /* count lines with them */
'| VAR VM', /* store count in variable VM */
'|?f:', /* third stream */
'| locate /VL/', /* locate vertical low */
'| count lines', /* count lines with them */
'| VAR VL' /* store count in variable VL */

Entitlement = 0*VL + 0.75*VM + VH

Say 'Estimated entitlement is' Entitlement
Problem Three: Determine Entitlement of a Logical Partition

/* Determine rough entitlement */
'PIPE (end ?)',
'| CP Q PROC TOPOLOGY',
'| f: fanout',
'| locate /VH/',
'| count lines',
'| VAR VH',
'?f:',
'| locate /VL/',
'| count lines',
'| VAR VL'

Entitlement = 0*VL + 0.75*VM + VH

Say 'Estimated entitlement is'
Entitlement

• No continuation after VAR VL, as that is end of Pipelines
Multistream Pipes - Lookup

Details

Master/Keys

Matching records

Unmatched Detail records

Unused Master/Keys

LOOKUP
Problem Four: Do I have the Service I need?

- Given a list of z/VM APARs, such as:
  - VM65942 VM65988 VM66071 VM65867 VM65865 VM65870

- How do I tell which, if any, are missing from my z/VM system? What if the list is even larger?
Problem Four: Do I have the Service I need?

- **Input streams**
  - Details will come from the CP command `QUERY CPSERVICE`
    - Example to get all APARs applied that start with VM6 use
      ```
      CP QUERY CPSERVICE APAR VM6*
      ```
    - Our list of required APARs will come from a file.

```
===== * * * Top of File * * *
===== VM65942 VM65988 VM66071 VM65867 VM65865 VM65870
===== * * * End of File * * *
```

- **Output streams**
  - Matching APARs are good and on system, keep in a file.
  - Other APARs are not of interest, so throw away
    - Use HOLE stage in Pipelines for illustrative purposes
  - Missing APARs keep in a file
Problem Four: Do I have the Service I need?

<table>
<thead>
<tr>
<th>QUERY CPSERVICE APAR VM6*</th>
</tr>
</thead>
<tbody>
<tr>
<td>APAR</td>
</tr>
<tr>
<td>VM65355</td>
</tr>
<tr>
<td>VM65481</td>
</tr>
<tr>
<td>VM65644</td>
</tr>
<tr>
<td>VM65741</td>
</tr>
<tr>
<td>VM65752</td>
</tr>
<tr>
<td>VM65846</td>
</tr>
<tr>
<td>VM65860</td>
</tr>
<tr>
<td>VM65865</td>
</tr>
<tr>
<td>VM65866</td>
</tr>
<tr>
<td>VM65870</td>
</tr>
<tr>
<td>VM65871</td>
</tr>
<tr>
<td>VM65872</td>
</tr>
<tr>
<td>VM65877</td>
</tr>
</tbody>
</table>

- Note the one header line
  - Will want to DROP that before entering Lookup
- APARs in columns 1-7
  - Will want to do lookup based on these columns
Problem Four: Getting List of Required APARs in Shape

--- * * * Top of File * * *

--- VM65942 VM65988 VM66071 VM65867 VM65865 VM65870

--- * * * End of File * * *

PIPE < required apars a | SPLIT | CONS
VM65942
VM65988
VM66071
VM65867
VM65865
VM65870

- Need one APAR per line/record
  - Use SPLIT APARs in columns 1-7
  - Will want to do lookup based on these columns
Problem Four: Do I have the Service I need?

/* Check for Service */

'PIPE (end ?)',

'CP QUERY CPSERVICE APAR VM6*', /* Get all service */

'| Drop 1', /* remove header */

'|l: lookup 1.7', /* APAR number is first 7 characters */

'| SORT UNIQUE 1-7', /* Remove Master */

'|> APPLIED APARS A', /* Applied APARs */

'|< required apars a', /* Read list of APARs */

'| SPLIT', /* Create one APAR per line */

'|l:', /* Secondary streams */

'| HOLE', /* Not matched from details, just ignore */

'?l:', /* Tertiary streams */

'|> MISSING APARS A' /* Masters that were not referenced */
Why the SORT UNIQUE 1-7?

- The output stream for matches includes the details and the master. We only care about the detail.
- There are other ways to accomplish this with other options on the LOOKUP stage.
You can write your own stages in REXX and other Languages

- The programs have a filetype of REXX.

- Basic construction is giant loop where you pull in data from the Pipe stage in front of you and write out data in the stream from you.

- For example, we have a file that has information in inches and we want to convert to centimeters.
  - Even SPEC doesn’t do this
  - We can write a simple program
REXX Stage for Inches to Centimeters

/* I2C REXX: Convert Inches to Centimeters */
DO FOREVER
  "READTO nextrec"
  IF rc <> 0 THEN LEAVE
  inches = nextrec
  centimeters = inches * 2.54
  "OUTPUT" inches centimeters
END
Exit

pipe literal 1.3 10 4 2 | split | i2c | cons
1.3 3.302
10 25.40
4 10.16
2 5.08
Ready;
CP System Services and Pipelines

- STARSYS stage - *ACCOUNT, *LOGREC, *SYMPTOM
- STARMON stage - *MONITOR
- STARMSG stage – Connect via IUCV to messages

Pipe starmon mondcss shared | locate 5 x01| locate 8 x0F| spec 21.8 1 77.8 nw 93.4
c2x nw | cons

MNTDASD2 ALEXIAA 1FFFFFFF
COYLE 3FFFFFFF
HOTTENMA 03FFFFFFF
MONWRITE 07FFFFFFF
BRAZIE 1FFFFFFF
RIVADENE 01FFFFFFF
OVVMCHEK 01FFFFFFF
MEAS00 01FFFFFFF
QWATCH 01FFFFFFF
Further Reading - Introduction

- CMS Pipelines home page  [http://vm.marist.edu/~pipeline](http://vm.marist.edu/~pipeline)
  - Papers by Melinda Varian
  - CMS Pipelines Tutorial

- CMSPI-P-L Mailing List
  - Subscribe through [listserv@vm.marist.edu](mailto:listserv@vm.marist.edu)

- CMS Pipelines Author’s Edition
  - Part 1. Introduction
  - Part 2. Task Oriented Guide

- z/VM CMS Pipelines User’s Guide

[rvdheij.wordpress.com](https://rvdheij.wordpress.com)
Summary

- Pipelines is powerful!
- Pipelines is useful!
- Pipelines is fun!